UNIT 19 MICROPROCESSOR INTERFACING & APPLICATIONS

Structure

19.1 INTRODUCTION

Although the microprocessor is very powerful and can carry out many functions, in order for it to be useful, it has to be built into a system, such as a microcomputer, along with other peripherals such as memory and input/output units. The technique of connecting these peripherals to the microprocessor is called **interfacing**. Interfacing peripherals to the microprocessor has to be studied in detail because it is not simply connecting these devices to the microprocessor pins that is involved. We have studied in Unit 17 that the microprocessor has only one set of address and data lines to be used by all the peripherals. This means that interfacing technique should include some sort of a scheme to select the peripheral device that is required to communicate with the microprocessor while at the same time isolating all other peripherals, even though all the peripherals stay connected to the microprocessor all the time.

Being versatile and programmable, microprocessors can be put to use in a variety of applications. The basic hardware is the same for all the applications. The input and output devices may, however, be different in each case based on the application needs. Different programmes are written for different applications and loaded into the program memory for execution by the microprocessor. In this Unit, we will discuss the issues involved in the design of a microprocessor based systems and study a typical application as an example.

Objectives

After a study of this unit, you should be able to

- explain the concepts of memory map, partial decoding, bus contention and wait states as applied to a microprocessor system,
- design memory and input/output systems for the 8085 microprocessor,
- understand advanced topics in input/output design such as interrupts, serial input/output and Direct Memory Access, and
- define and formulate applications of the 8085 microprocessor.

19.2 MEMORY MAP

Memory map is a graphical representation of the total available memory in a microprocessor system and how it is used. It is not always that a microprocessor system

requires the entire amount of memory that it possibly can have. Many practical systems require only a fraction of the maximum memory size of the microprocessor. However, since a microprocessor is a universal device meant to be used in a wide range of applications, the manufacturers provide as large a memory addressing capability as possible within their design constraints.

The 8085 microprocessor has 16 address lines which give a maximum possible memory size of 64 K bytes. Let us assume that we have a microcomputer built using the 8085 which has 2 K bytes of ROM and 256 bytes of RAM. Figure 19.1 shows the memory map of the scheme.

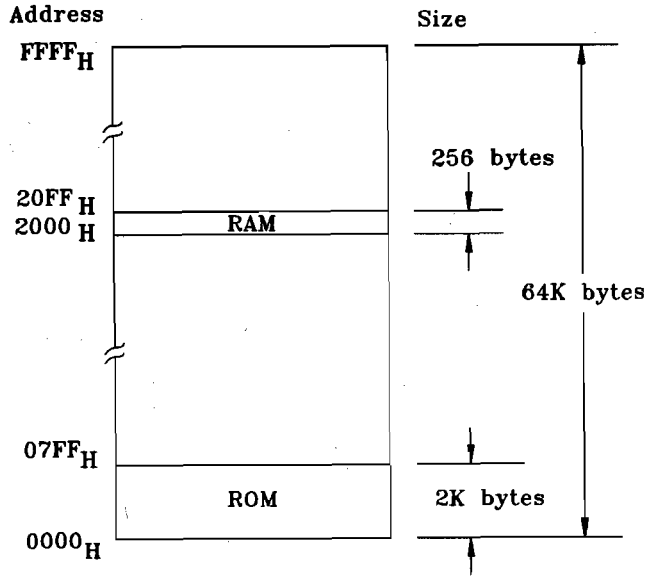


Figure 19.1 : Memory map of a 8085-based microcomputer

Note that all memory addresses are expressed as hexadecimal numbers. The maximum possible size of 64 K has the address range from 0000_H to $FFFF_H$. The 2 K ROM occupies the address range 0000 to $07FF_H$. 256 bytes of RAM occupy the address space from 2000_H to $20FF_H$.

Example 19.1

Find the ending address of an 8 K-byte memory if the starting address is '0'

Solution

8 K-bytes = 8192 bytes

Starting address is 0. Ending address is 8191

$0_{10} = 0_H$; $8191_{10} = 1FFF_H$. The address range is 0000_H to $1FFF_H$.

Example 19.2

What is the ending address of a 2 K-bytes memory whose starting address is 3000_H?

Solution

2 K-bytes = 2048 bytes = 800_H bytes

Ending address is $3000_H + 800_H - 1 = 37FF_H$

19.3 MEMORY INTERFACE DESIGN

Designing a memory interface for a microprocessor consists of the following steps: We first decide the size of the memory required and the address range it should occupy (memory map). We then choose the type of the memory device to be used based on cost and other conditions such as speed, availability etc. Based on this information, we now know how many chips of the chosen device are needed to give the required memory size. Then, we design the circuitry that will enable the memory to communicate with the CPU whenever the CPU initiates either a read or a write cycle, and disables it at other times. If the memory consists of more than one chip, only the appropriate chip or chips should be enabled. Occasionally, the design has to be modified to take into account the timing considerations, i.e., the memory should respond to the CPU neither too quickly nor too late.

19.3.1 Memory Address Decoding

Memory address decoding circuit utilises the address lines of the microprocessor to generate the signals that enable the appropriate memory chips as required. The chip select function of the memory device is used for this purpose, as mentioned in Unit 16. The decoding circuit may use either a decoder or gates or a combination of both.

Let us assume that we want to design a memory circuit of size 2 K-bytes of RAM to be interfaced to an 8085 microprocessor. Let us choose the address for this memory from 0000_H to $07FF_H$. Let us assume that we have a single memory chip of size 2 K-bytes at our disposal. To address 2 K-bytes, we need 11 address lines. Of the 16 address lines A_0 to A_{15} of the 8085, we will use the lines A_0 to A_{10} for this purpose. This will give all the address combinations from 000_H to $7FF_H$ and we will be able to access any of the locations in the 2 K-byte RAM. However, we should not leave the five higher address lines A_{11} to A_{15} unused. These lines should all be 0s in order to uniquely define the address range 0000_H to $07FF_H$. For any other combination of values on these lines, other address ranges in the overall 64 K memory may be selected. The operation of utilising these higher address lines to uniquely enable only the required memory address and exclude other addresses is called **memory decoding** and the circuit which does this is called the memory decoding circuit Figure 19.2 shows the memory decoding circuit for this example. Notice that $\overline{CS}$ goes low (and thus enables the memory) only when the lines A_{11} to A_{15} are all 0s. That is, this circuit

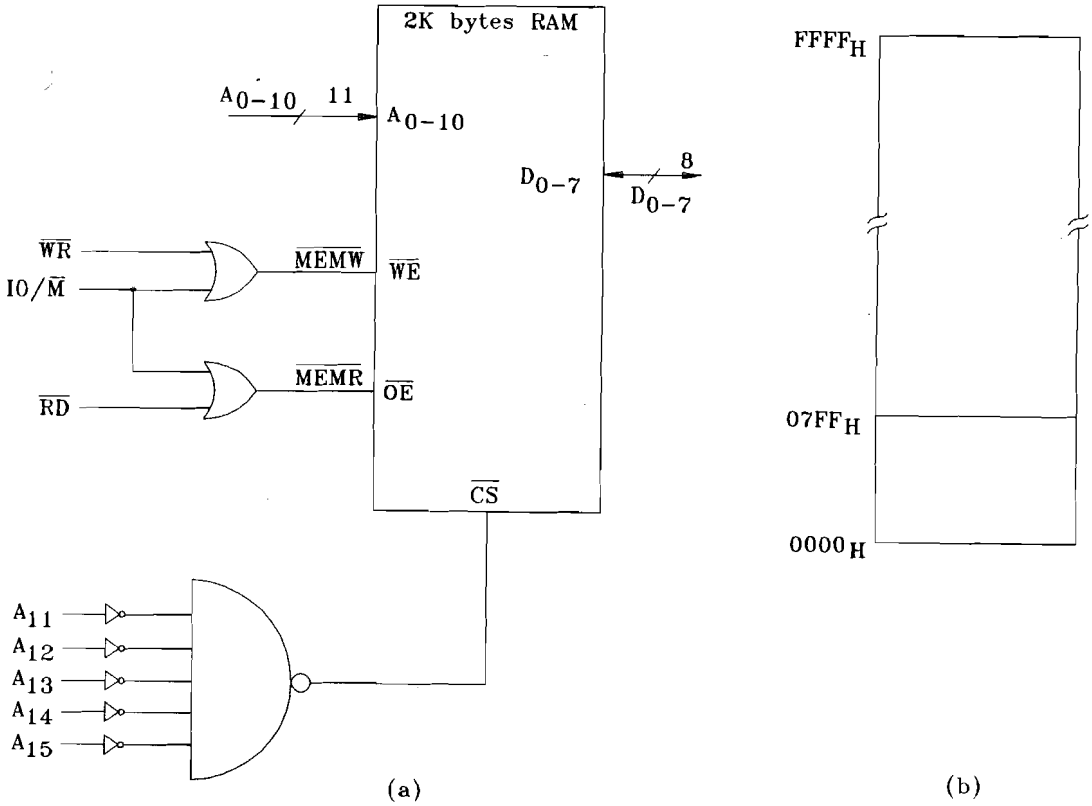


Figure 19.2 : (a) A 2 K-byte RAM circuit for 8085 microprocessor, (b) Memory map

will respond only to addresses 0000_H to $07FF_H$ and to no other combinations. $\overline{OE}$ is the output enable signal and the memory can be read from only when this is active low. When IO/$\overline{M}$ and $\overline{RD}$ of the 8085 are both low, it indicates a memory read cycle and the OR gate output ($\overline{MEMR}$) is low which enables $\overline{OE}$. Similarly, for a memory write cycle IO/$\overline{M}$ and $\overline{WR}$ are low and the corresponding OR gate output ($\overline{MEMW}$) goes low which enables the memory write operation.

If the memory were a ROM instead of a RAM, then there would be no $\overline{WE}$ and hence no need for $\overline{MEMW}$ signal and the corresponding OR gate. Also, data lines D_{0-7} would be unidirectional, only coming out of the chip.

Several variations of this design are possible. The required addresses may not always form the lowest possible address block in the memory map. There may be several memory units in a system occupying different regions of the memory map as shown for example, in Figure 19.1. Let us now consider an example of a memory design with the required addresses somewhere in the middle of the memory map.

Example 19.3

Design a memory circuit to interface a 4 K-byte RAM to an 8085 microprocessor with starting address 7000_H.

Solution

4 K-byte = 4096 bytes or 1000_H bytes

If the starting of the memory is 7000_H, its ending address should be $7000_H + 1000_H - 1 = 7FFF_H$.

The required address range is from 7000_H to $7FFF_H$.

Memory of size 4 K-bytes will have 12 address lines i.e., address lines A_0 to A_{11} will go to the memory chip to address 4 K-bytes.

The upper address lines A_{12} to A_{15} should take on values such that the memory address will be 7000_H to $7FFF_H$. i.e., A_{12} to A_{15} should take values 0111 respectively. For other combination of values, the memory will not be accessed. Remaining part of the design is identical to that of the previous example. The circuit is shown in Figure 19.3.

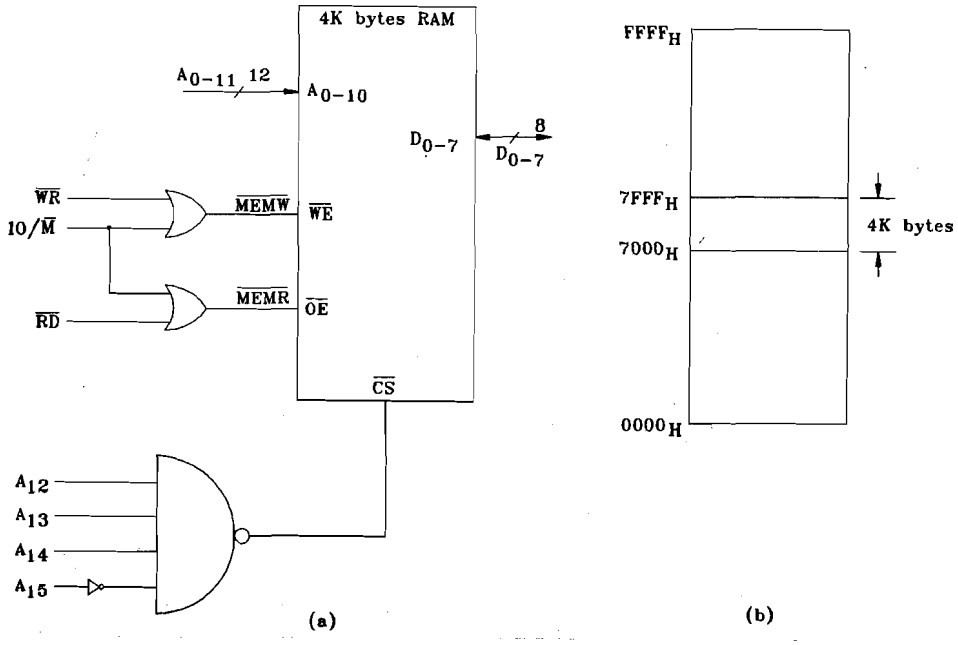


Figure 19.3 : (a) Circuit of the 4 K-byte RAM for 8085 (b) Memory map

The two examples we have seen so far have both used a single chip in the design. Sometimes, we may not get a single chip large enough to cover the entire memory range we want to have. In such a case $\overline{CS}$ signal should be generated separately for each chip. The required memory range is partitioned into subranges and each of the chips is enabled for the appropriate subrange. Let us understand this concept by means of an example.

Example 19.4

Design a memory system of size 8 K-bytes to interface with the 8085 microprocessor. Start with the address 0000_H. Assume that you have to use four pieces of 2 K-bytes RAM.

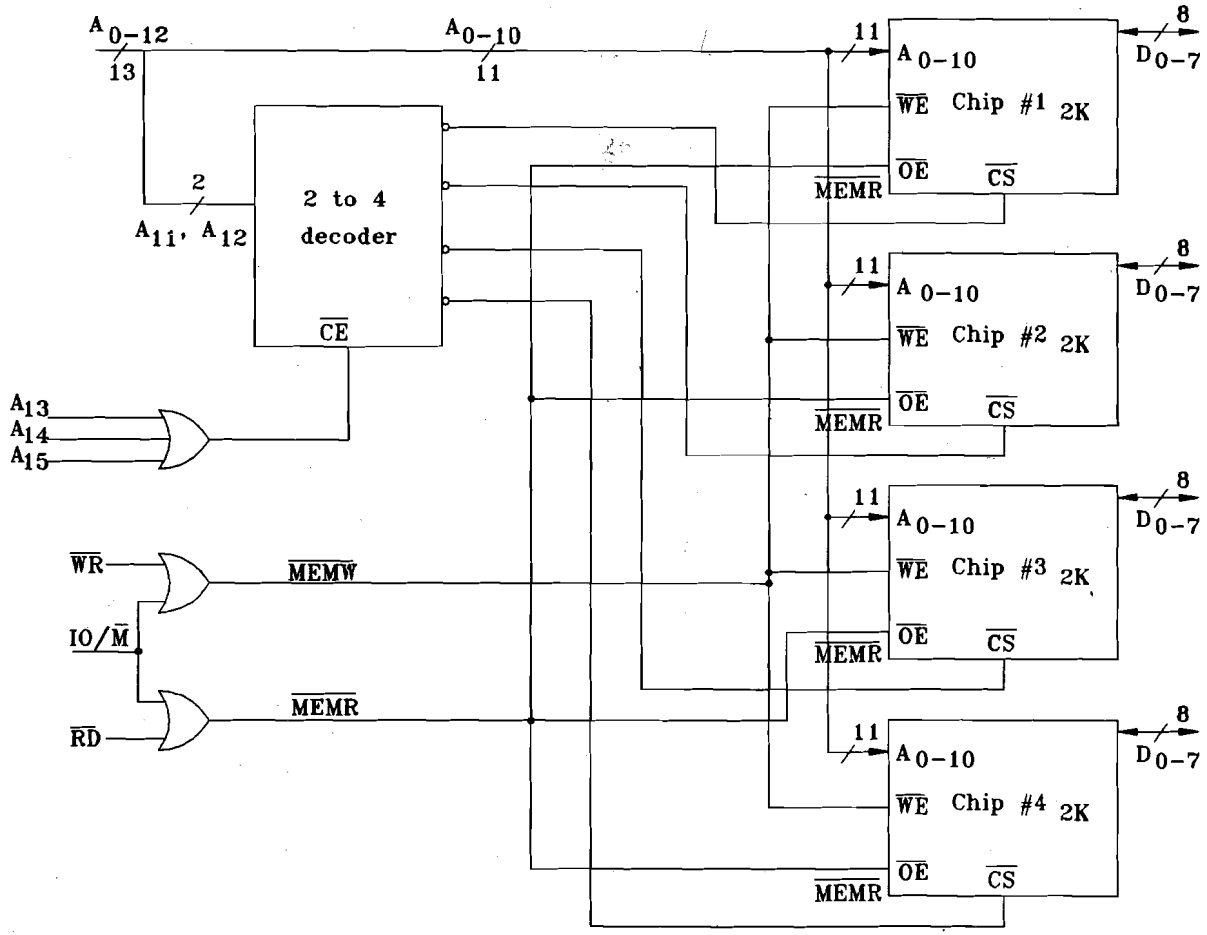
Solution

8 K-bytes = 8192_{10} bytes = 2000_H bytes

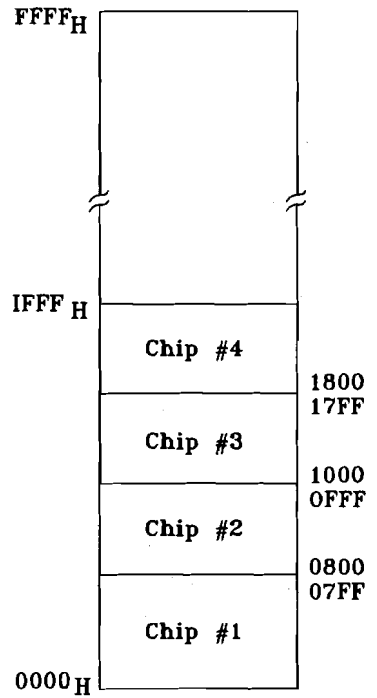
If the starting address is 0000_H, ending address is $1FFF_H$.

Each RAM has 2 K-bytes of memory. We can divide the total range into four subranges as follows:

0000 - 07FF	Chip # 1
0800 - 0FFF	Chip # 2
1000 - 17FF	Chip # 3
1800 - 1FFF	Chip # 4



(a)



(b)

Figure 19.4 (a) : 8 K - byte RAM interface to 8085 microprocessor (b) Memory map

Out of a total of 13 bits required to address 8 K memory, 11 bits are needed by each chip to select one of 2 K address locations contained in it. The remaining two bits will then be used to select one of the four chips to address the required subrange. Figure 19.4 shows the circuit. Note that the address lines A_{11} and A_{12} are decoded into four chips. Each output corresponds to a combination of A_{11} and A_{12} as required to address one of the subranges as given below:

A_{12}	A_{11}	*Subrange*
0	0	0000 - 07FF
0	1	0800 - 0FFF
1	0	1000 - 17FF
1	1	1800 - 1FFF

The three remaining address lines A_{13}, A_{14} and A_{15} are required to be all low to address the total range properly. The three-input OR gate ensures that only if they all are low, the address decoder will be enabled thereby enabling the memory circuit. For any other combination of A_{13}, A_{14}, A_{15} (which means that the address is outside the required range of 0000 - 1FFF), the memory circuit is totally disabled.

SAQ 1

Using the required number of 1 K x 4 bits RAM, design a memory circuit of size 2 K-bytes with the starting address 7000_H to interface with the 8085.

19.3.2 Partial Decoding

When memory requirement of a microprocessor system is small, we can save in the hardware needed for the memory circuit design by simplifying the decoder design. Instead of using all the higher address lines not used by the memory chips to generate the chip select signals, we can leave some or all of them unused. The result is that when the CPU sends an address on the address bus in order to access a particular memory location, only the address lines which are used are matched to select the required location. This location may not be unique since with the same address combination, several memory locations can exist with different combinations of values for the unused portion of the address bus. However, this is not a problem since only one actual memory exists and only this will respond to the CPU address. At other valid locations there are no physical memory devices and hence there is no question of multiple responses from these locations affecting the data transfer. This type of decoding used in memory circuit design is called Partial Decoding since the memory address lines are only partially decoded (only some are used, others are not used). The usefulness of partial decoding lies in the simplified decoding scheme resulting in reduced and therefore less expensive hardware. The drawback of course, is that only systems with small memory requirement can use this scheme since we can not use all the multiple address locations corresponding to the unused address lines. Let us understand partial decoding with an example.

Let us assume that we need to build a memory for the 8085 microprocessor system with 2 K-bytes of ROM and 1 K bytes of RAM. There is no other memory in the system. We are not particular about the addresses for ROM and RAM and the only restriction, of course, is that they should be mutually exclusive.

Let us fix the address of ROM to be from 0000_H to $07FF_H$ and the address of RAM to be 0800_H to $0BFF_H$. ROM requires 11 address lines A_0 to A_{10} while the RAM requires only 10 address lines A_0 to A_9. From the two address ranges we notice that address bit A_{11} is 0 for the ROM address range and it is 1 for the RAM range. This can be used to have these ranges mutually exclusive by connecting $\overline{A_{11}}$ to $\overline{CS}$ of RAM and A_{11} to $\overline{CS}$ of ROM. Address lines A_{12} to A_{16} are not used in this design. Figure 19.5 shows the circuit and the memory map.

In deciding the addresses for the two memory chips, we have assumed that the address lines

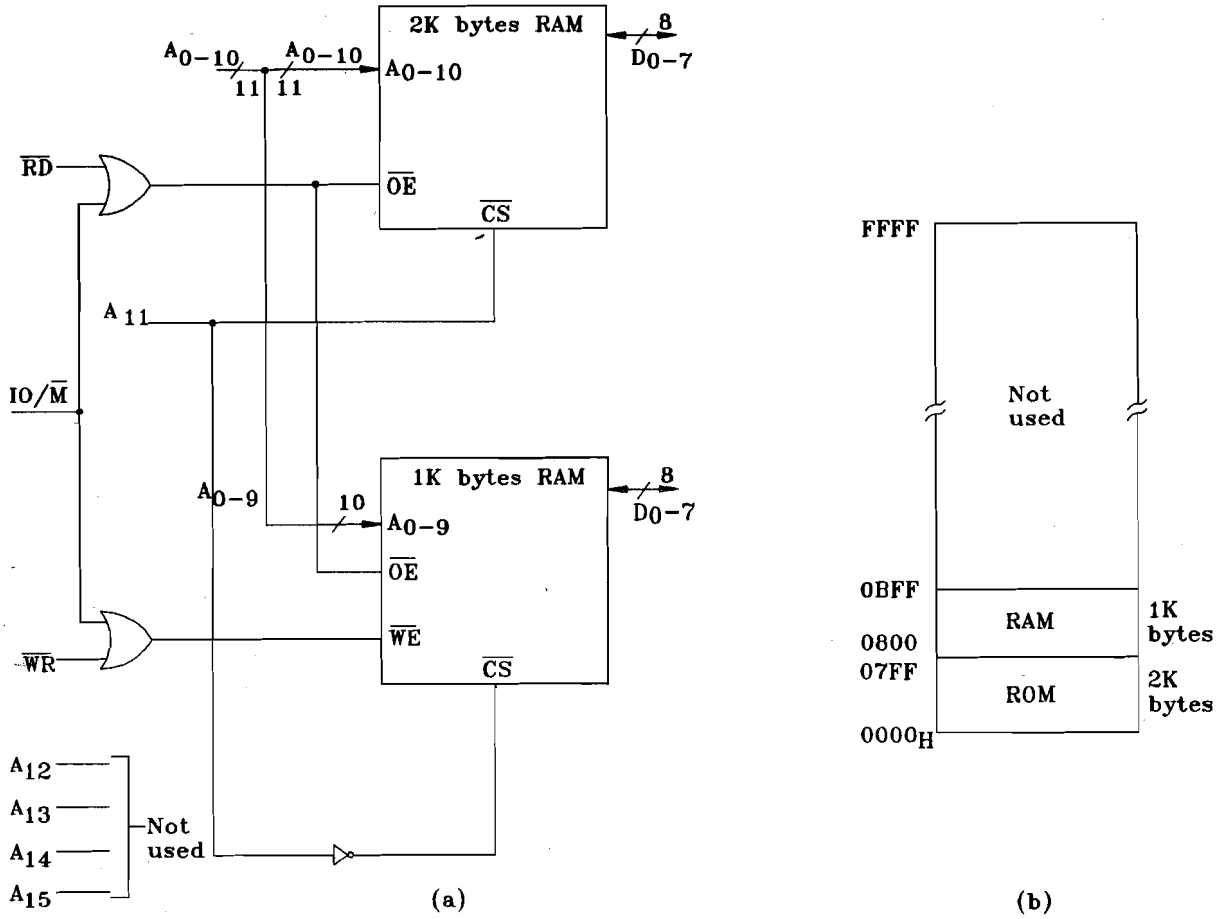


Figure 19.5 : A partially decoded memory system (a) Circuit (b) Memory map

A_{12} to A_{15} are all 0s. This is not a restriction. For any other combination of these address lines also, the same two memory chips will be accessed because these lines are not used and only the address lines A_0 to A_{11} will be used for address decoding of the memory. For example, if the address ranges are from F000 to F7FF and F800 to FBFF, still the same two memory chips will be accessed by the circuit.

Example 19.5

Find the range of addresses for the memory chip in the circuit shown in Figure 19.6.

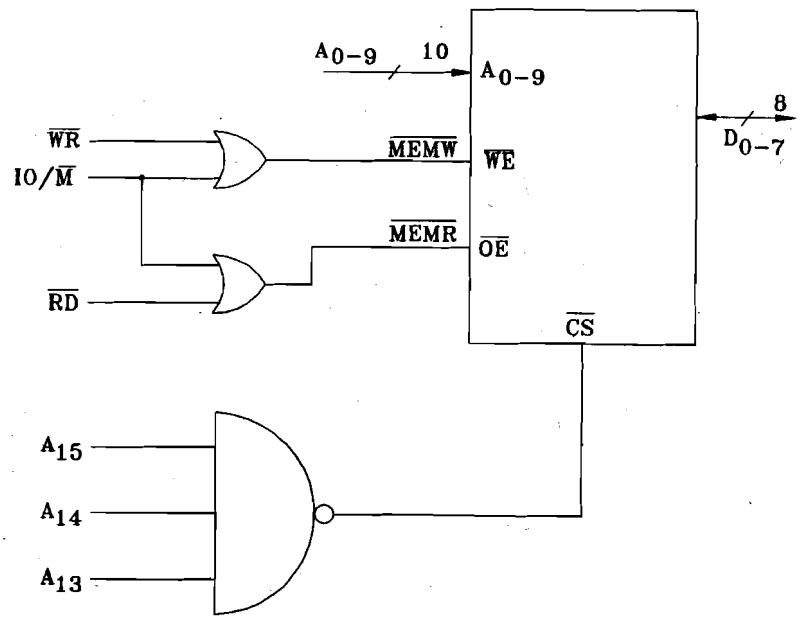


Figure 19.6 : The partially decoded memory system for Example 19.5

Solution

The size of memory in the chip is 1 K byte since it uses 10 address lines plus 8 data lines. 1 K locations have addresses in the range 0000_H to $03FF_H$. In order to find the actual address of the memory in this circuit, we have to consider A_{13}, A_{14}, A_{15} which are all 1. Since A_{12}, A_{11} and A_{10} are not used, we can assume them to be 0s. The addresses are therefore

$$1110\,00\ \{\ \begin{matrix} 00 & 0000 & 0000 \\ \\ 11 & 1111 & 1111 \end{matrix}\ \}$$

which translate to $E000_H$

to $E3FF_H$

This range was arrived at with the assumption that A_{12}, A_{11} and A_{10} are 0s. For other combinations, we will get different ranges, but they are all equivalent (as far as this circuit is concerned) since they will all represent the addresses in the same memory device. For example, the combination 100 for A_{12}, A_{11}, A_{10} will give the address range from $F000_H$ to $F3FF_H$.

19.3.3 Bus Contention and Wait States

Memory devices take time to respond. After an address is placed on its address lines and the memory is enabled, the memory takes time to internally decode the address and become ready either to place the requested data on the data lines (in the case of read cycle) or to read the data placed on the data lines for storage (in the case of a write cycle). This time delay is called the *memory access time* and is made up of several delay components. Furthermore, the access times are different for read and write operations. We shall not be going into the details of how these delay times come about, but only say that when we use a memory device in a microprocessor system, we have to make sure that there is a match between the speed requirements of the microprocessor as given by its read and write cycle times and the access time specifications of the memory device.

With the present day technology, memory devices are made extremely fast and they are available in a wide range of sizes, speeds and cost. Especially for a rather slow microprocessor such as 8085, there should be no difficulty in identifying memory devices with the required specifications. However, it may be of interest to learn about two problems related to timing that are likely to arise in a memory interface design. Again, in the design of a simple 8085 system, we can avoid these problems by choosing proper memory devices, but it is worthwhile to know about these anyway.

Bus contention

A bus contention occurs when two devices try to use a bus at the same time. In the case of 8085 microprocessor, we know that the same set of eight lines are used as lower byte address lines A_0 to A_7 as well as the data lines D_0 to D_7. Signal ALE is used to define the function of these lines at any given time. When ALE is high, these lines are used as low order address bus. Refer to Figure 19.7.

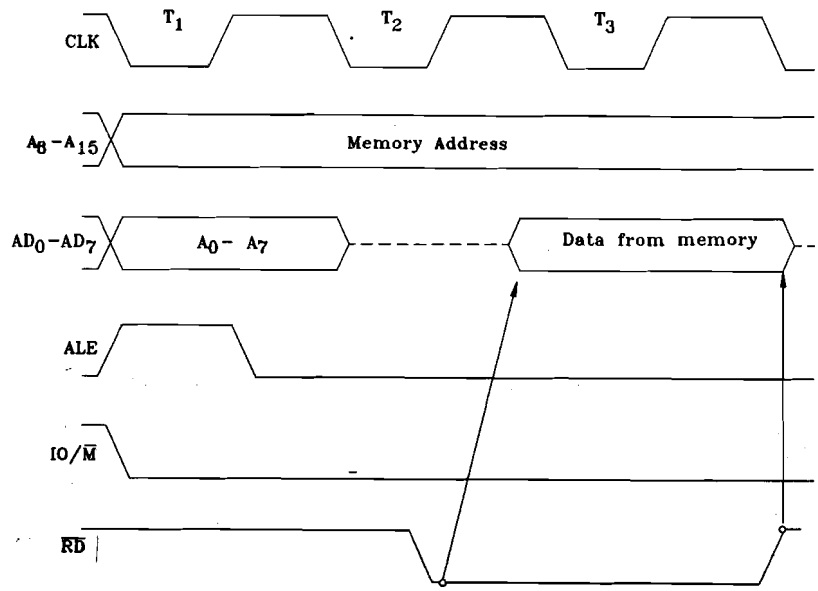


Figure 19.7 : Timing diagram of a memory read cycle

It is quite possible, if we have memory devices which are extremely fast to respond, that in a memory read cycle, as soon as the address is out, memory may respond and try to place the data at its output. However, since the same set of eight lines are used both as low order address and data bus, it is possible that the address information is still on this bus and has not yet been removed by the CPU and if the memory tries to place data on the bus which still carries the address, a contention results. Contention causes not only unreliable and useless data, but it may also damage the hardware by driving currents above the specified limits because of the addition of two signals on these lines.

Bus contention can be easily avoided in the 8085 system by proper design. Referring to Figure 19.7, we see that the $\overline{RD}$ signal is generated only in the beginning of the second clock cycle. By this time, ALE has become low and the address information has been removed from the lines AD_0 -AD_7. By enabling the output of the memory using the $\overline{RD}$ signal, we can avoid bus contention. Even if the memory is so fast as to try to respond as soon as the address is made available in the beginning of T_1, the data output from the memory chip is possible only if its $\overline{OE}$ signal is activated and this can not happen until $\overline{RD}$ is generated in the beginning of T_2 at which time the bus is free to be used as the data bus.

You may notice that in all the memory circuits that we have designed we have always used $\overline{MEMR}$ (which is the OR combination of $\overline{RD}$ and IO/$\overline{M}$) as the signal to enable the output of the memory chips.

Wait states

The other extreme of bus contention is the possibility that the memory devices may be so slow to respond to read or write requests that the CPU may not be able to complete read or write cycles successfully. In the case of a read cycle, the CPU expects data from memory on the data bus during the cycle T_3 and in the case of a write cycle the data from CPU to be stored in memory is available until the end of the clock cycle T_3 before which time the CPU expects that it will be written into the memory. This is also when $\overline{WR}$ signal is deactivated. If we use memory chips with access times longer than those specified by the requirements as explained above, CPU can not successfully complete read or write operations.

The ideal solution to this problem is of course to use faster memory devices which match the CPU speeds required. Assuming that such devices are either not available or too expensive, we have another solution : wait states.

Wait state is a dummy clock cycle that the CPU introduces between the second and third clock cycles (T_2 and T_3) of a read or a write cycle to give a slow memory device additional time to respond. Wait state has no specific activity and it is an idle clock cycle. Status of all the signals remain as they were during T_2 clock cycle. Wait state gives the memory additional time equal to the period of the clock cycle to either come up with the data required by the CPU in a read cycle or accept and store the data provided by the CPU in a write cycle. Furthermore, there is no restriction that only one wait state may be introduced in a read or write cycle. As many wait states as are required until the memory is ready to respond will be generated by the CPU, thus giving the memory additional time equal to multiples of clock periods. All the wait states are placed between the regular clock periods T_2 and T_3.

How does a CPU know which devices require wait states and how many wait states to generate? There is a signal input called READY in the 8085 microprocessor. If this input is high, read and write cycles are performed normally with only three clock cycles. However if this input is made low, a wait state is introduced after the T_2 cycle of every read or write cycle. As long as this signal remains low, additional wait states continue to be generated. Only when this input is restored to high, does the CPU continue with the T_3 cycle and complete the memory read or write operation. Whenever we find a mismatch in the speeds of one particular memory and the CPU specifications, we have to build additional hardware to provide an active low signal to this READY input during every read or write operation of the particular memory chip. The duration for which this signal should be active will be decided by how many wait states are required which in turn depends on the amount of mismatch between the memory speed and the CPU requirement. Designing such a hardware is beyond the scope of this course material.

19.4 INPUT/OUTPUT PORTS

In a microprocessor system, memory holds program and data and the CPU runs the program using the data and produces results. These results are also sometimes stored back in the memory.

However, we need to collect data to start with and constantly update it as conditions change. Similarly, we need to send the results to external devices for them to act on these results. This suggests that in addition to the memory, microprocessor systems, require external units through which inputs can be given and outputs taken. Since there are a large number of possible input devices such as switches, keyboard, light pen etc. and output devices such as Cathode Ray Terminals, lights, relays etc., it is impossible to design interfaces to match the physical characteristics of every possible input/output device. We therefore standardize the design of input and output interfaces by using a concept known as input/output ports (I/O Ports).

Port is an intermediate circuit used to interconnect an input or an output device to a microprocessor. You can imagine a port to be analogous to a person who receives incoming letters in an office before sending them to the proper person for action. In the context of output, it can be thought of as analogous to the person who sends out letters on which actions have already been taken by different persons in the office. In other words, a port acts as receiving and shipping point of data in and out of the microprocessor.

I/O ports have two functions. The times at which CPU either requires data or gives out data may not be the same as when the devices are ready to either give or receive data. One of the functions of the I/O ports is to take care of this. When an input device has data which needs to be processed, the CPU may not require it as it may be busy with some other task. The processor needs data from any input device only when it encounters the corresponding instruction during the program execution. It is, therefore, very difficult for the input device to make data available exactly at the instant when it is required by the CPU. Whenever an input device has a data to be processed, it sends it to the input port associated with the device and it is held there until the microprocessor is ready to receive and process it. Similarly an output device may not be ready to receive data from the microprocessor whenever the CPU executes instructions to output results to that device. The result is sent to the output port corresponding to this device and is held there until the device is ready to receive and use it.

The other function of I/O ports is to isolate the physical devices from the CPU whenever they are not communicating with the CPU. In memory design, we saw that each chip has a chip select input which is used to either select or deselect the chip depending on the memory address that the CPU is trying to access. Similarly, a large number of I/O devices are connected to the data bus of the microprocessor to be used at different times. However, when the CPU tries to access any one of them, others should not respond. Since physical devices like switches and relays come with varying descriptions, not all of them may have features similar to the chip select feature for the purpose of enabling/disabling them. By standardizing the interconnection of different I/O devices through the use of ports, we can solve this problem. Ports will use ICs which will have chip select or output enable functions.

19.4.1 I/O Device Select Pulses

When several input/output devices are used in a system, at any one time the microprocessor either reads or writes to the one that is required at that time by the program being executed currently. In order to make this possible, devices are given addresses similar to the addressing scheme in memory. 8085 can support upto 256 input and another 256 output devices. In hexadecimal notation, this is the range from 00_H to FF_H for each function. Notice that this range requires only an 8-bit address and hence only eight of the sixteen 8085 address lines i.e., A_0-A_7, will be used for decoding input/output ports. Figure 19.8 shows the I/O map of the 8085 microprocessor.

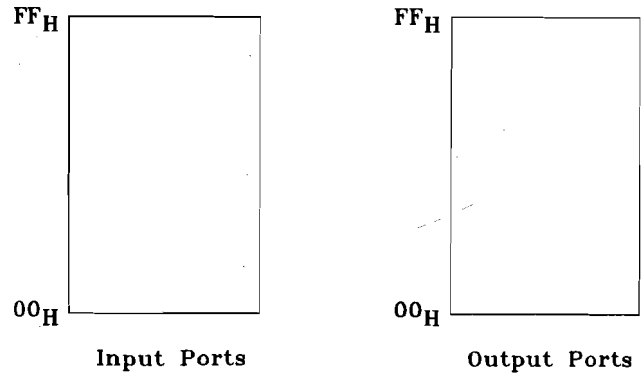


Figure 19.8 : I/O map of 8085 microprocessor

I/O decoding is similar to memory decoding in the sense that the address lines (in this case A_0-A_7) are used to select the port just as address lines $A_{0\text{-}15}$ are used to select the memory location. However, there is a difference. In memory design, a single decoder circuit decodes a range of memory addresses since memory chips contain a large number of address locations. On the other hand, I/O decoding has to be done for individual devices since these are independent of each other and each has its own address. The second difference : I/O decoding circuit uses the address of the device, $\overline{RD}$ or $\overline{WR}$ signals (depending on whether it is an input or an output port) and the IO/$\overline{M}$ signal (which will be high when CPU performs an I/O operation as against low for memory operation) and comes out with a signal called device select pulse. This pulse is used to select (or enable) the I/O port required to communicate with the CPU. Since each device is identified with a decoding circuit and an associated device select pulse, only the device of choice will be enabled and all others will be disabled at that time. Figure 19.9 explains this feature.

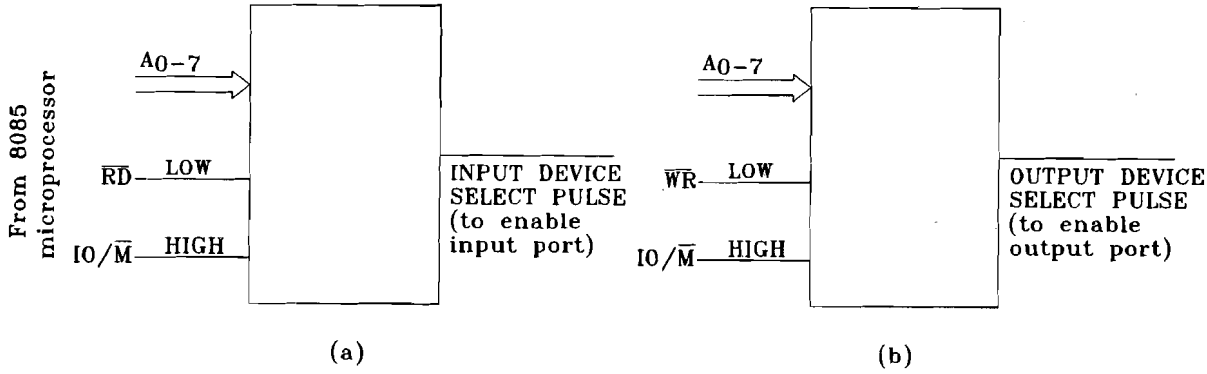


Figure 19.9 : Device select pulse generation (a) for input port (b) for output port

19.4.2 I/O Interface Design

Let us consider the example of interfacing eight switches designated S_0, S_1, S_2 .....S_7 to the 8085 microprocessor. These switches should act as an input port with port address 20_H. In the ON position, the switch should provide an input of 1 and in the OFF position, a 0.

The design will consist of three parts: switches, the port and the port decoding circuit. These are shown in Figures 19.10(a), (b) and (c) respectively.

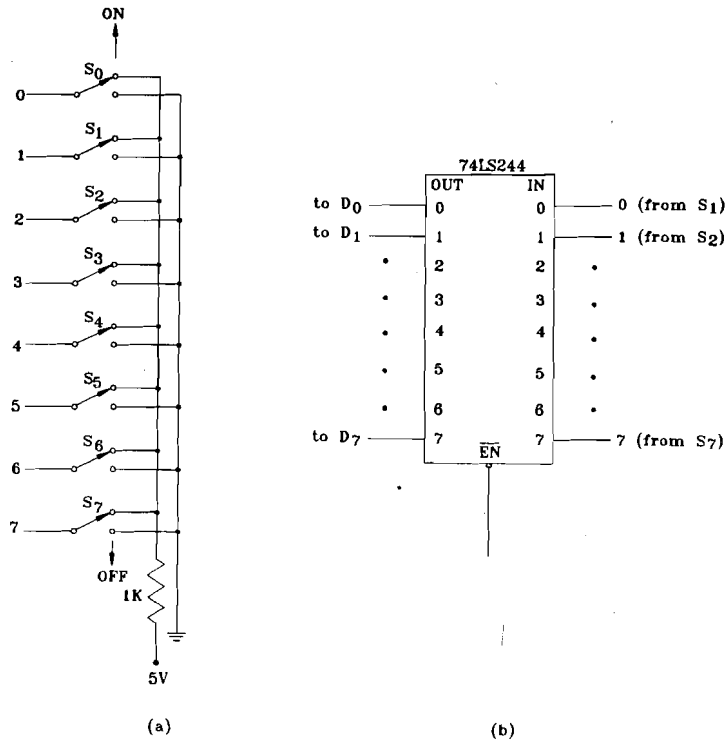


Figure 19.10 : I/O interface design : (a) Switches (b) Input port

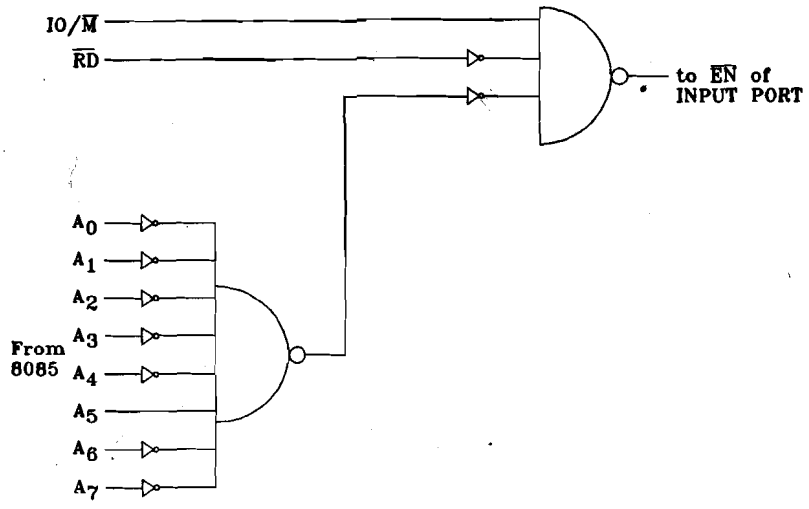


Figure 19.10 : (c) Port decoding circuit for input port

As can be seen from Figure 19.10(a) switches produce 0 V or 5 V as outputs depending on their respective positions. These outputs are given to the inputs of the 74 LS 244 buffer as shown in Figure 19.10(b). They are connected to the data bus D_0-D_7 of the 8085 microprocessor when the chip is enabled. Chip enable is in turn caused by the decoding circuit of Figure 19.10(c) which responds in this case to the port address 20_H along with IO/$\overline{M}$ high and $\overline{RD}$ low corresponding to the input read bus cycle. Figure 19.11 shows the complete input device interface.

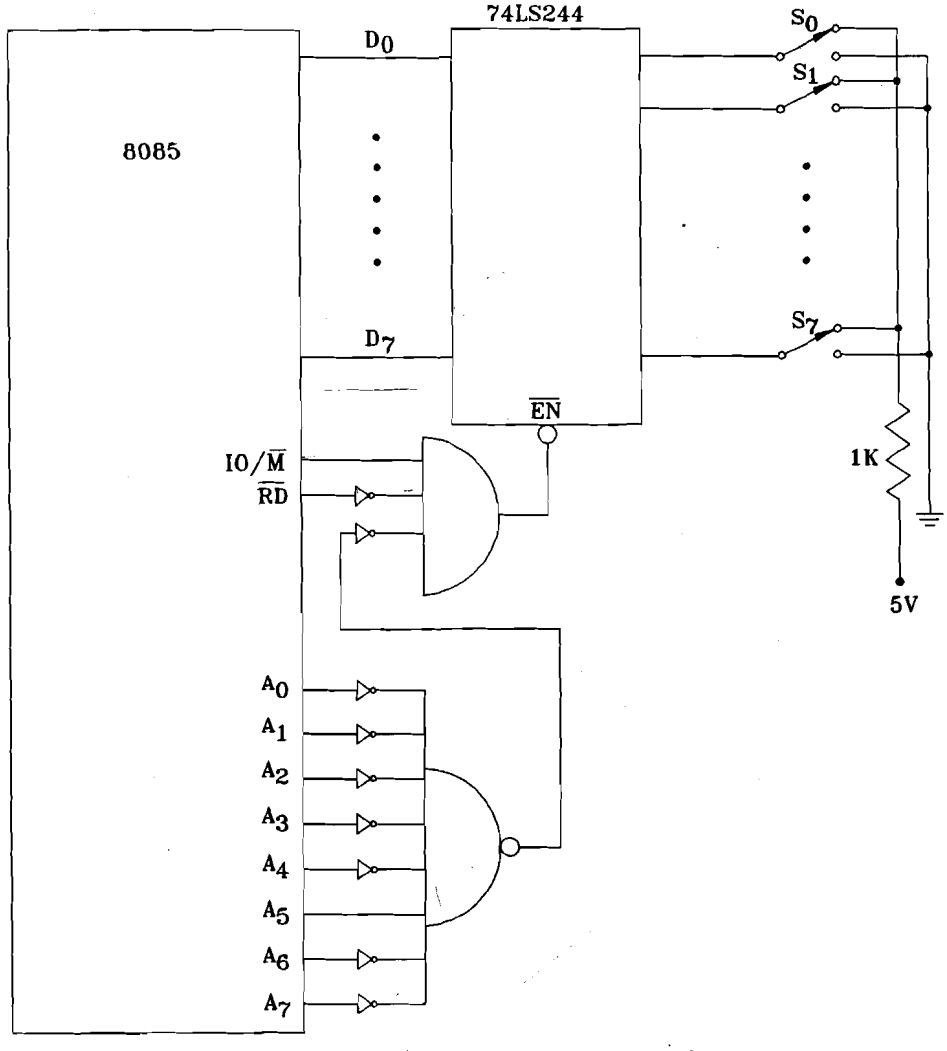


Figure 19.11 : Complete input device interface for the 8085 microprocessor

Output device interface design is similar to the input design with one difference, however. The decoding circuitry and the device connections are as in the input interface, but the port will have an extra feature. In the case of input design, port consists of buffers which are enabled by the device select pulse, but in the case of output design the port should contain latches in addition to buffers. Latches are clocked by the output device select pulse. Data on the microprocessor data bus meant for the port is latched on to the port when microprocessor generates data meant for the particular output device. The port holds this data in a latch and passes it on to the output device when the output buffers are enabled. Figures 19.12(a) (b) and (c) show the device interconnection, port design and the port decoding circuit respectively for an output device interface design with port address 20_H. The devices are LEDs driven by the port with a 1 turning the LED ON and a 0 turning it OFF.

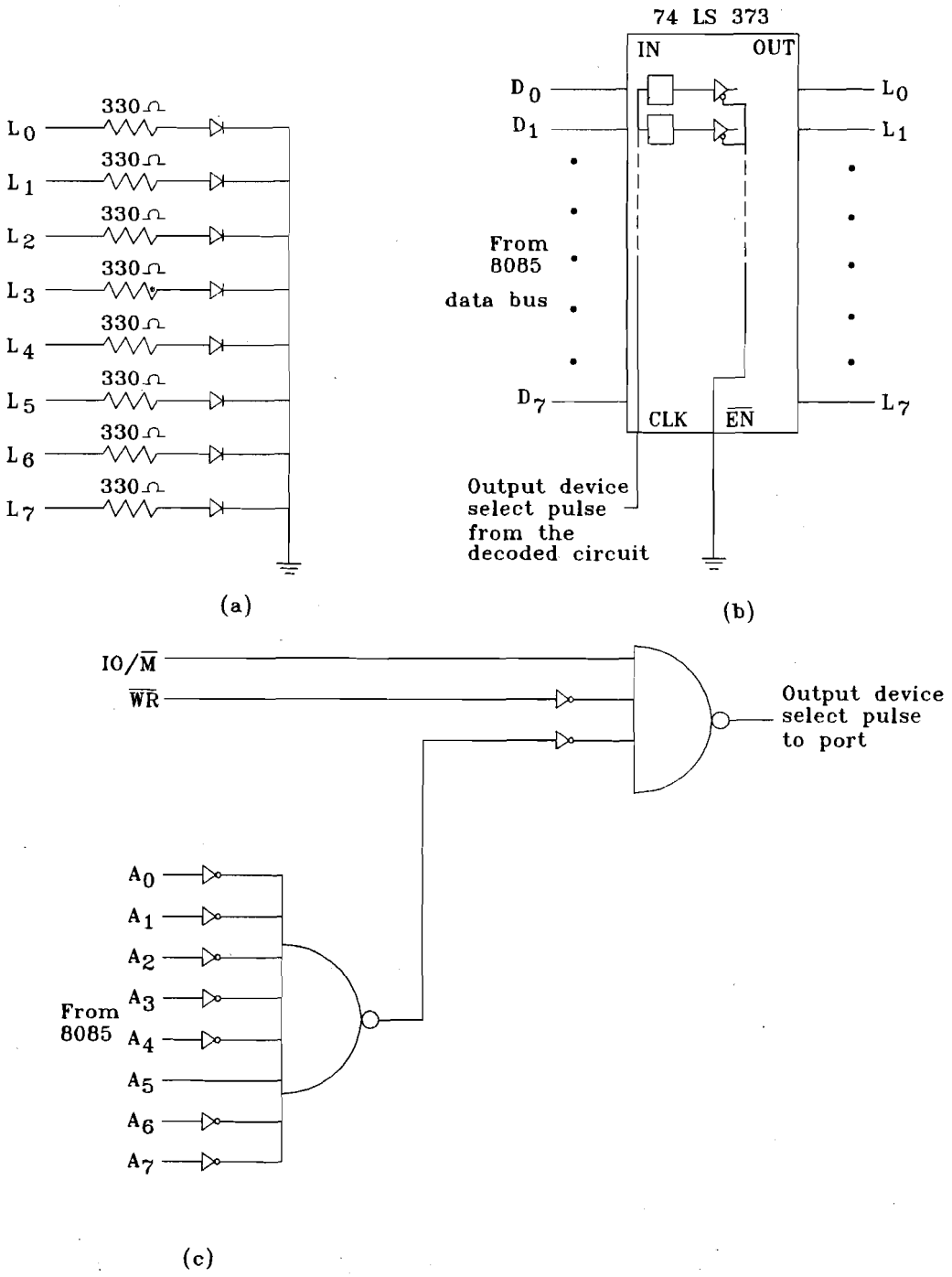


Figure 19.12 (a) Output device (b) Output port (c) Output decoding circuit

SAQ 2

Design an output port with port address 80_H to interface a pair of BCD displays to an 8085 microprocessor.

19.4.3 Memory Mapped I/O

The 8085 microprocessor has a memory size of 64 K bytes, 256 input ports and 256 output ports. As we have seen in the previous sections, these spaces are exclusive of each other and will be accessed by the microprocessor by means of control signals $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$ and $\overline{IOW}$. However, in small systems requiring only a limited memory size and a small number of input/output ports, it is not necessary to go through extensive designs separately for memory and input/output ports. It is possible to locate input/output ports at the unused memory addresses and thereby reduce the design effort by having to design only a common hardware for both memory and input/output ports. Such an approach to the design of input/output ports is known as memory mapped I/O.

The advantage of memory mapped I/O over conventional I/O (also known as isolated I/O) is the reduction of hardware required for decoding circuitry. The microprocessor sees I/O ports as memory addresses, but in the actual implementation, memory does not exist in these addresses, but buffers/latches which constitute I/O ports. Since memory addresses are used to represent these memory mapped I/O ports, memory instructions should also be used to transfer data from and to these ports. This has the advantage of flexibility and speed since there are a large number of available data transfer instructions from memory as against only two (IN, OUT) from I/O ports. Furthermore, memory reference instructions can access any CPU register whereas the I/O instruction can access only the accumulator.

The only disadvantage of memory mapped I/O is that the size of available memory is reduced by the extent of its use by I/O ports, but this is not a real problem in most systems which do not require the full 64 K bytes of memory space.

19.5 ADDITIONAL I/O TECHNIQUES

In the simple I/O design examples of the previous section, we have assumed that the input/output devices are generally ready for data transfer whenever the microprocessor desires or requires such a transfer. In practice, however, there may be situations when I/O devices have to initiate the data transfer based on data availability or on the readiness of the device to accept data. Let us take the example of a printer connected as an output device to a microprocessor. When a file stored in the memory needs to be printed, the microprocessor can not dump its entire content on to the printer. This will result in loss of data from the file if the speed at which the printer prints a character is slower than the speed at which the microprocessor outputs the characters and if the printer does not have an internal buffer memory to store the characters as they are received. The solution is to establish a procedure called **handshake** by which the microprocessor and the I/O device will communicate to make sure that the device is indeed ready for the data transfer and that no data is either missed or transferred more than once. In the case of the printer, for example, the printer can give out a signal called BUSY which the microprocessor can monitor. If BUSY is low, it means that the printer is free to receive the next character and the microprocessor can send the same. If, on the other hand the BUSY signal is high, it indicates that the printer has not yet completed printing the previous character and the mircroprocessor will do well to wait to send the next character in order to ensure proper printing. It is of course, the responsibility of the microprocessor to check the status of I/O devices and to initiate data transfer at appropriate times.

19.5.1 Interrupts

The process of monitoring handshake signals from I/O devices, consumes a large part of the program execution time of the microprocessor. Furthermore, sometimes, the processor gets unnecessarily held up with a device which does not generate the requisite handshaking

signals due to a local fault. Interrupt is a mechanism of I/O data transfer which ascertains the readiness of the I/O device for the data transfer while at the same time eliminating the hold up of the CPU for monitoring I/O devices.

In Interrupt I/O, the CPU carries out a main program until an I/O device desires a data transfer. When it happens, viz., when an input device has new data to be passed on to the CPU or an output device requires fresh data, the concerned device interrupts the CPU by means of a signal called *Interrupt Request*. When this signal is received, the CPU completes the execution of the instruction that is being processed currently and sends a return signal to the interrupt device called *Interrupt Acknowledge*. At this point, the CPU suspends the execution of the main program and executes a subroutine called *Interrupt Service Routine* which takes care of the I/O transfer that the interrupting device required in the first place. On completion of this subroutine, the CPU returns to the main program and continues its execution from where it left. This way, any I/O device requiring data transfer gets prompt service, the validity and correctness of data is assured and the CPU does not spend a whole lot of time waiting for an I/O device to get ready to send or receive data.

The microprocessor handles an *Interrupt Service Routine (ISR)* exactly in the same way as it handles a Subroutine call. There is however, one difference. Subroutine calls are placed by the program at the required places in the main program and contain the memory addresses at which the subroutines reside. On the other hand, Interrupt Request can occur at any time and ISR calls can not be placed in the main program by the programmer. This means that ISR addresses to which the CPU should go to execute these routines can not be supplied as part of the main program and hence have to be predetermined at the time of the design of the microprocessor itself.

8085 Interrupts

8085 has five interrupt inputs (as can be seen from Figure 17.3). Four of these are automatically transferred to specific locations of the memory when they occur. These are

Interrupt	*Call Location*
TRAP	0024_H
RST 7.5	$003C_H$
RST 6.5	0034_H
RST 5.5	$002C_H$

They have the same order of priority as shown in the table. For example, if RST 7.5 and RST 5.5 interrupt signals occur at the same time, the microprocessor will first attend to RST 7.5 and only then to RST 5.5. Out of these interrupts, TRAP is called a *Non Maskable Interrupt* (NMI) because while the microprocessor can be programmed to ignore all the other interrupts when they occur, TRAP can not be masked and will always have to be attended to by the microprocessor. This is usually reserved for emergency situations such as power supply failure or fire alarm in system design. If no such provision is required in a system, it is better left unused. Masking and unmasking (or disabling and enabling) the interrupts other than TRAP are carried out by the 8085 instruction DI and EI respectively. Instruction DI disables all 8085 interrupts (other than TRAP) and EI enables all of them. It is also possible to selectively mask and unmask the RST 7.5, RST 6.5 and RST 5.5 interrupts by the 8085 instruction SIM. We shall not go into the details of this instruction here.

Interrupt request is passed on to the 8085 microprocessor by a signal at the appropriate input of the microprocessor. For TRAP and RST 7.5 interrupts, a positive going edge is enough to cause the request and subsequent processing of the interrupt. These two interrupts are called edge triggered interrupts. On the other hand, all other 8085 interrupts are called level triggered interrupts. For these interrupts to be recognized and serviced, the devices generating these interrupts have to present a HIGH signal at the appropriate input and keep this signal HIGH as long as it takes the 8085 to recognize and service the interrupts. This means that with TRAP and RST 7.5, a device requesting interrupt will always be serviced whereas with the other interrupt types, it will be serviced only if the device keeps the request alive until it is attended to.

The 8085 interrupt with the lowest priority is INTR. Eight different types of INTR interrupt are possible. On an INTR request by a device being acknowledged by the microprocessor, the type number (0 to 7) is indicated to the microprocessor by a specially designed hardware for this purpose. Each of the interrupt types has its own predetermined memory address at which the corresponding Interrupt Service Routine resides and based on the type

information supplied by the hardware, the CPU directs the program counter to the appropriate location to execute the corresponding interrupt service routine.

Typical uses of interrupt are in data acquisition, monitoring critical parameters, data display and refresh, and real time clocks that run continuously even as programs are executed in the processor.

19.5.2 Serial I/O

8085 is an 8-bit microprocessor. This means that its data bus and registers are 8 bits wide and that memory and I/O are organized as 8 bit words. CPU sends data to and receives data from the peripherals 8 bits at a time, one bit on each line of the data bus. Such a transfer which is the normal mode of data transfer between the processor and its peripherals is known as parallel I/O since all the bits corresponding to each data byte are transferred in parallel. Occasionally, though, it may not be possible to send (or receive) all the bits of a data byte at once; they have to be sent (or received) one bit at a time. Such a data transfer in which bits of a data byte are sent (or received) serially one after the other is known as serial I/O.

Why do we have to resort to serial I/O? When the microprocessor has to send data to a peripheral which is located at a far off place, it is uneconomical to run wires between the microprocessor and the peripheral corresponding to each line of the data bus. We would rather run a pair of wires (one for data and one ground wire) between the two points and use it to transmit all the data bits one after the other. This provides a considerable amount of saving in the cost of wires, but will also result in considerable slowing down of the data transfer. It is always a compromise between the cost and speed of data transfer that has to be worked out before deciding the mode of transfer to be either parallel or serial. Serial I/O is a particularly attractive alternative when the data transfer is between computers located at different places such as in a computer network. A typical example is the computerized railway reservation system operating from various railway stations spread all over the country. Surely, it is uneconomical to provide parallel data line connections between each one of these computers all across the country and serial I/O is the only practical option for such a situation. What is even more attractive is that even the single pair of wires required for serial I/O need not be run afresh as the telephone wires already existing between all these stations can be made to carry the serial data. However, the electrical characteristics of the data generated by computers and those of the telephone signals are different and in order for the telephone lines to be used for serial I/O for computers, the computer data at the sending and receiving ends should be modified to match the electrical characteristics of the telephone lines. This is accomplished by devices known as **Modems** which are provided at both ends of telephone line used for data communication between two computers.

Data rates of serial I/O are defined by the term *baud*, which represents the number of bits of data transferred in a second. Standard baud rates are 300, 1200, 2400, 9600, 19200. The actual data rate chosen changes from application to application and will depend on factors such as the capabilities of the computers, the modem and the electrical characteristics of the transmission wires. The 8085 has two pins exclusively meant for serial I/O. These are designated SID and SOD. SID is the pin through which serial data is received by the microprocessor and SOD is the pin through which serial data is sent out by the microprocessor to a serial peripheral device. There are special instructions SIM and RIM respectively to send and receive serial data from and into the microprocessor.

19.5.3 Direct Memory Access

One other technique of data transfer between the microprocessor and I/O devices is known as Direct Memory Access, or DMA for short. Sometimes we need to transfer large quantities of data between microprocessor memory and I/O devices. Examples are situations when data is acquired from an input device to be stored in memory for later processing or when a file stored in memory is to be printed on a printer. In such instances, it is desirable to complete the transfer of data between the memory and the concerned I/O device as quickly as possible to reduce the delay in execution of whatever program the microprocessor happens to be running at that time. The saving in microprocessor time and consequent fast transfer is possible if the CPU is not required to participate in the data transfer mechanism and the I/O peripheral in question is directly allowed to access the memory to read or write, without having to go through the microprocessor as is generally the case. Such a technique of data transfer is called the Direct Memory Access (DMA) since I/O peripherals access the memory directly, without going through the microprocessor.

DMA is resorted to quickly load programs from secondary storage devices such as floppy disks to main memory before executing a program and is similarly used to output large volumes of data for display or printout.

In the 8085 microprocessor, DMA works as follows: There is an input designated as HOLD in the 8085 microprocessor. An external I/O device requiring a Direct Memory Access sends a request signal through this input. When the CPU receives a signal at its HOLD input, it suspends its external bus activity and releases the address, data and control buses to the peripheral requesting the DMA. The availability of the bus is indicated to the peripheral by the CPU by means of HLDA (or Hold Acknowledge) signal. On receiving HLDA, the peripheral takes control of the bus and completes the desired data transfer at the end of which it releases the HOLD request. When the HOLD request is removed by the peripheral, CPU responds by withdrawing the acknowledgment HLDA, once again takes control of the bus and proceeds with the execution of the program which was temporarily suspended. DMA increases the speed of data transfer upto four times compared to the regular method of data transfer.

In order to effect a smooth interface and implementation of the DMA, a peripheral device known as DMA Controller is available. Its functions include transmitting the DMA request from the I/O device to the CPU, passing the HLDA back to the peripheral and initiating and terminating the actual DMA operation. It also keeps track (with program control by the microprocessor) of the memory addresses from or to which data is to be transferred and the number of data bytes to be transferred. Upto four I/O peripherals can be serviced by a single DMA controller.

19.6 MICROPROCESSOR APPLICATIONS

Being a very versatile and low cost device, the microprocessor can be put to use in a variety of applications in many areas. The type of applications to which a microprocessor can be put is limited only by the user's imagination. However, it is important to know how a microprocessor-based system is designed and how different hardware and software subsystems are put together.

Following are the steps in the design and implementation of a microprocessor-based system:

1. Problem specification,
2. Hardware design,
3. Software design,
4. Development of software
5. Hardware/software integration
6. System test

These steps are illustrated by means of an example.

19.6.1 A Peak and RMS Meter for Periodic Signals.

Problem specification

The objective of this application is the complete development of a stand-alone microprocessor based system which will measure, compute and display the peak value and root-mean-square (rms) value of a given periodic voltage waveform.

The completed system is required to :

1. sample the given periodic signal,
2. digitize the sampled values through a microprocessor controlled A/D converter,
3. input the digitized values to the 8085 microprocessor,
4. compute the peak value and the rms value of the waveform, and
5. display the peak and rms values.

The system is to be designed and constructed from the chip level.

Hardware design

The hardware may be considered in two sections: (a) the microcomputer consisting of the 8085 microprocessor, RAM, ROM and I/O ports and (b) the external circuitry used to sample, digitize, and display.

8085 based microcomputer

Figure 19.13 shows the microcomputer circuit. It consists of an 8085 microprocessor, an 8156 RAM I/O and an 8755 EPROM I/O chips. These chips constitute the core of the microcomputer. The 8156 chip contains 256 bytes of RAM and three I/O ports. The 8755 chip contains 2 K-bytes of ROM and two 8-bit I/O ports. The advantage of using these chips is that they contain the decode logic for I/O ports. When the memory contained in these chips are properly decoded with required addresses, the I/O ports also get decoded with corresponding addresses. Furthermore, each I/O port contained in these chips can be used either as an input port or as an output port under program control. The following control signal lines are connected to all three devices.

ALE	:	latches address lines
$\overline{RD}$	:	used to read from device
$\overline{WR}$	:	used to write to device
IO/$\overline{M}$	:	used to designate either memory operation or I/O port operation
CLK	:	clock output of 8085
RESET	:	generated by 8085 in response to external hardware; will reset all I/O ports to input

The following additional specifications pertain to the 8085.

HOLD	:	connected to ground to disable
INTR	:	- do -
RST5.5	:	- do -
RST7.5	:	- do -
$\overline{RESET\ IN}$	:	will generate RESET signal and force program counter to 0000; connected external hardware to initiate program. (Pin 36 of 8085)
RST6.5	:	vectored interrupt ; connected to applications hardware, vector location is 0034_H.

not indicated in the Figure 19.13

The eight data/address lines (AD_0-AD_7) are bused between three devices. They are used to transmit the 8 data bits and the 8 low order address bits between the processor and the memory devices. Address lines A_8-A_{10} are connected between the 8085 and 8755. Since the 8156 RAM possesses only 256 bytes of memory (as opposed to 8755), it does not require these additional address bits.

RAM (8156) and ROM (8755) are mapped by address bit A_{11}. The 8755's ROM program resides in memory location 0000_H - $07FF_H$ (the extent of its 2-K byte memory), and the chip is selected when A_{11} is LOW. Hence, the A_{11} line is connected to the $\overline{CE}$ pin of the 8755. In contrast, the 8156 RAM chip is selected when A_{11} is HIGH, thus giving it the memory range 0800_H - $08FF_H$ (256 bytes).

Output ports *A* and *B* of the 8156 interface to the external hardware. Port *B* is designated as an output from 8085 and issues control bits to the conversion hardware. Port *A* is designated as an input and receives the 8-bit data word from the A/D converter.

I/O is performed by standard I/O techniques. When executing an IN or OUT instruction, the 8085 deselects memory and selects I/O ports by bringing IO/$\overline{M}$ HIGH and transmitting the 8-bit port address on address lines AD_0-AD_7 and A_8 - A_{15}. Since the 8156 is enabled when A_{11} is HIGH, the software designer must be certain to enable that bit in the port address.

On the 8156 chip, bits AD_0 - AD_7 control the specific port accessed.

AD_2	AD_1	AD_0	
0	0	0	Command/status register
0	0	1	Port *A*
0	1	0	Port *B*

Command/status register is a register which is part of all the programmable I/O chips. It is an additional port accessible to the microprocessor. By programming this register, we get different configurations for the I/O ports in the chip. (For example, in this application, Port *A* acts as input and Port *B* acts as an output port).

Address bit A_{11} translates to address bit AD_3 in the port address. Therefore, AD_3 will be HIGH for all 8156 I/O port addresses:

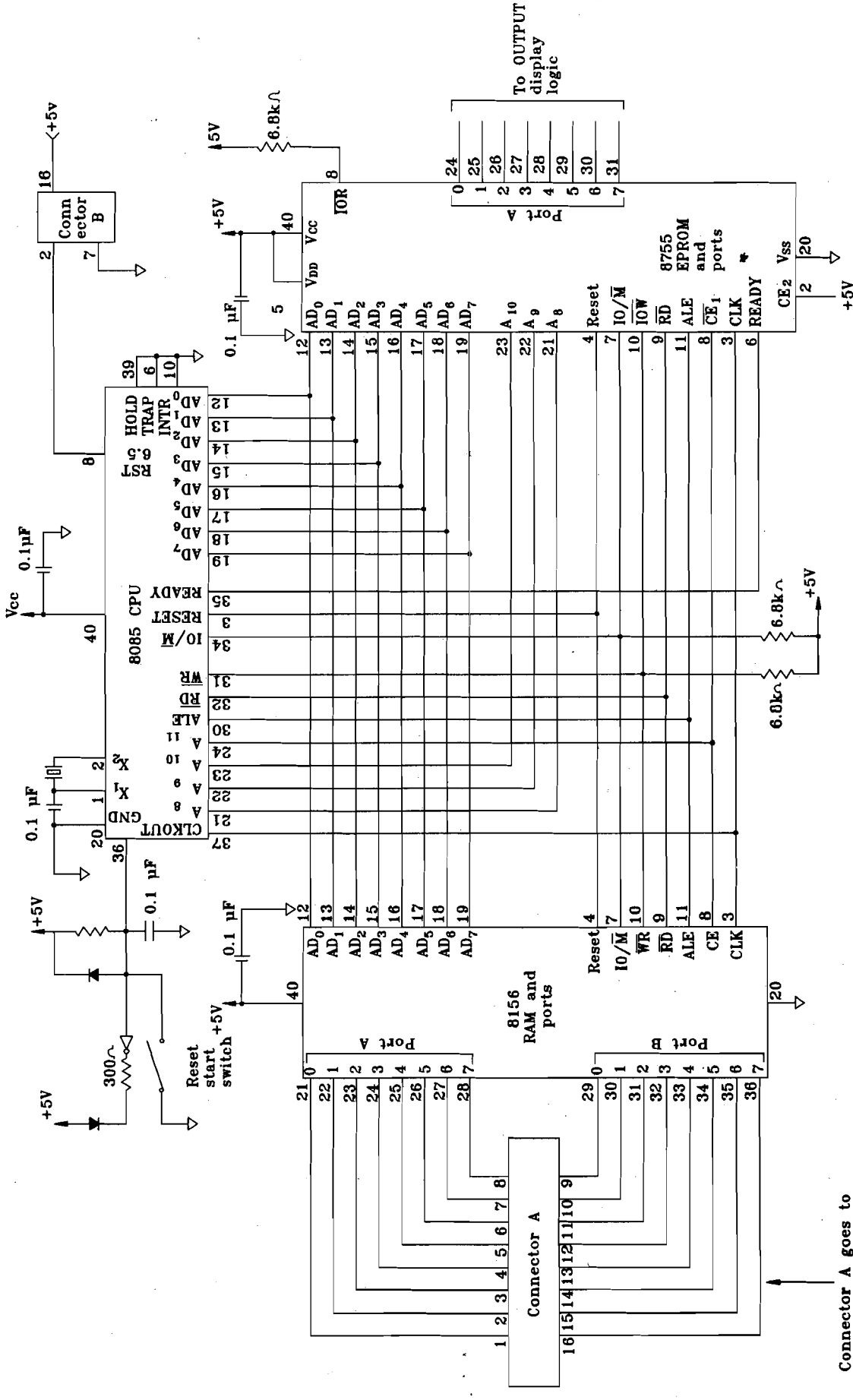


Figure 19.13 : Microcomputer system schematic

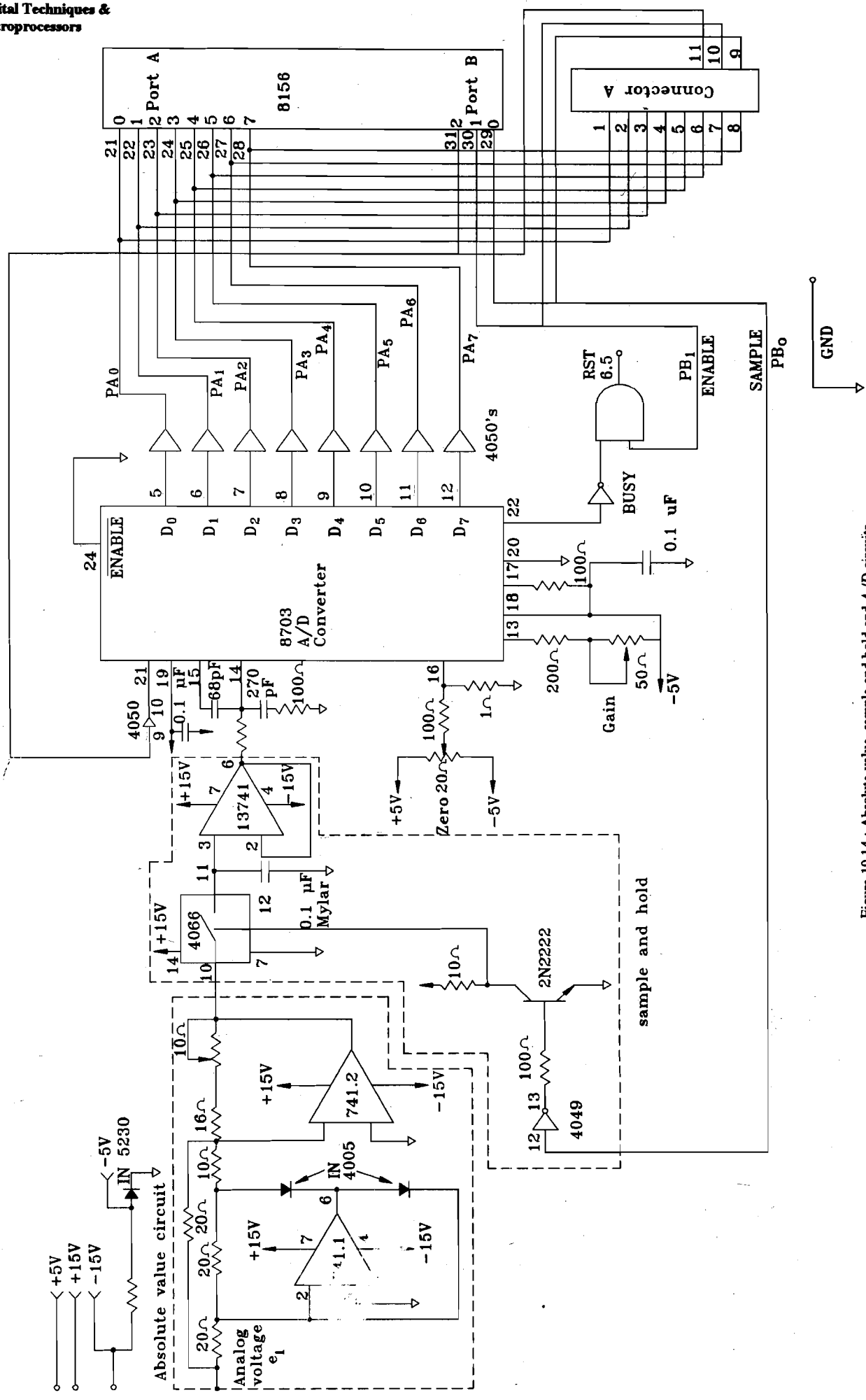


Figure 19.14 : Absolute value, sample and hold and *A /D* circuits

Port Address	AD_3	AD_2	AD_1	AD_0	
08_H	1	0	0	0	Command/ status register
09_H	1	0	0	1	Port *A*
$0A_H$	1	0	1	0	Port *B*

Four 7-segment displays are used to display the results, two each for the peak and rms values. The same port (port *A* of the 8755 chip) is used to activate all the four displays in turn. Each 7- segment display requires four bits of data. When a digit is to be outputted, the lower four bits are used for data and the upper four bits to enable one of the four displays. Each display has its own latch/hexadecimal 7-segment decoder combination (9356). When one of the bits of the upper four bits is enabled, the corresponding latch/decoder latches the lower four bits which is meant to be the data for that particular digit. The bits are decoded into the 7-segment display information and is used to light up the corresponding segments of the display. The process is repeated for each digit in turn.

Each bit of the 8755 port *A* must be defined as an output by a Data Direction Register (DDRA). Also, writing into either port *A* or DDRA requires that the 8755 device be enabled. Therefore, A_{11} (corresponding to AD_3 in an OUT operation) must be LOW. The resulting port addresses for the 8755 are :

AD_3	AD_2	AD_1	AD_0	Port Address	
0	0	0	0	00_H	Port *A*
0	0	1	0	02_H	DDRA

Besides the three Intel devices, the microcomputer includes a 1 MHz crystal oscillator to provide a master clock, a RESET switch to reset the system, and latched LED displays to output the peak and rms values.

A/D Conversion Hardware

The circuitry is composed of three circuits: absolute value, sample and hold, and A/D converter.

The absolute value circuit used rectifies the input voltage within the feedback loop of an operational amplifier (see Figure 19.14). When the input voltage e_1 becomes negative, the output of operational amplifier 1 becomes positive by one diode drop and turns off the upper diode. No effective current flows from operational amplifier 1, and the output voltage of operational amplifier 2 is $-e_1 = \text{abs}(e_1)$. When e_1 becomes positive, the lower diode is turned off, and the output of operational amplifier 1 becomes $(-e_1 - 0.7\ \text{V})$. The output of operational amplifier 2 then becomes $(-(-2e_1 + e_1)) = e_1$. In either case, the output voltage of operational amplifier 2 is the absolute value of the input wave, and no diode drop error is introduced. This is a variation of the precision rectifier circuit which you studied in Unit 13.

After the input voltage is rectified, it is sampled by the sample and hold circuit. A SAMPLE signal of 650 μs duration is transmitted from I/O port *B* of the 8156 and inverted by a 4049 CMOS buffer. When SAMPLE is HIGH, the output of the buffer is LOW, and the 2N2222 transistor is turned off. Since it is not conducting, its collector voltage remains at 15 V, and the 4066 analog switch is closed. To open the switch, SAMPLE is set LOW, the 2N2222 transistor saturates, and the gate voltage of the analog switch is effectively set to 0 V. When this analog switch is closed, a capacitor charges to sample voltage.

Finally, the sampled and rectified voltage is converted to an 8- bit digital word by the 8703 A/D converter. The 8703 is biased to provide a 0-10 V scale range (i.e., 00_H is equivalent to 0 V, FF_H to 10 V). All signals to the I/O ports of the microcomputer are buffered through 4050's to isolate the expensive CMOS A/D converters.

The eight digital output bits are connected to port *A* of the 8156 and are read during the interrupt service routine. Conversion is begun by a programmed START CONVERSION pulse transmitted from the 8085 through port *B* of the 8156. After the A/D converter has completed conversion of the input voltage, the status line $\overline{\text{BUSY}}$ goes LOW. When enabled by the software-controlled ENABLE bit, the inverted signal BUSY triggers the RST6.5 interrupt, and the interrupt service routine reads the valid data.

Since the BUSY output from the 8703 remains LOW ($\overline{BUSY}$ remains HIGH) between conversions, a software latch must be used to prevent further interrupts. Therefore, $\overline{BUSY}$ is ANDed with ENABLE to provide an RST6.5 interrupt only when the microcomputer is ready to accept it. During the interrupt service routine, ENABLE is set LOW and remains in that state until the next sample is taken and the next A/D conversion begun. Figure 19.15 depicts the timing of the control signals.

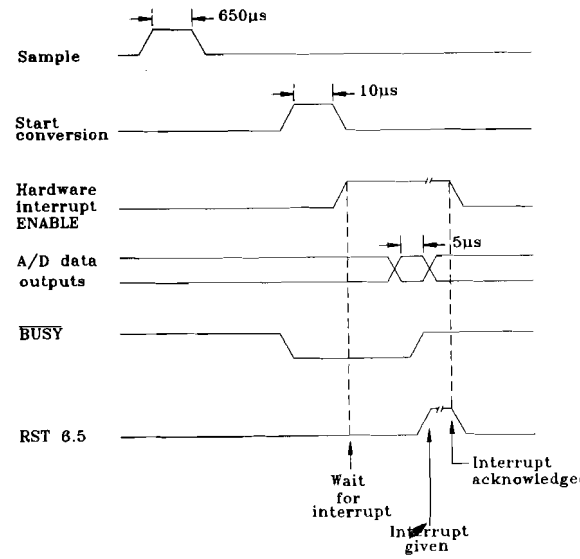


Figure 19.15 : Hardware Timing

Software

The software used in this application exists as a single program, assembled as an integral unit. It performs three primary functions: sampling of data, servicing interrupts, and computing the peak and rms values. Each function may be considered separately.

When the system is reset by manually enabling the RESET line of the 8085, the program counter is automatically forced to 0000_H. A JMP instruction at this address sends the processor to the beginning of the sampling and acquisition sequence (0037_H). This portion of the program enables the RST6.5 interrupt, and defines the 8156 I/O ports (port *A* = input, port *B* = output).

Next, the program turns on the sample bit (bit 0 of 8156 port *B*) for 650 µs by using a timing loop. It then disables SAMPLE and issues a START CONVERSION command (bit 2), after which it enables the software interrupt latch (bit 1). Finally, it reaches a HLT instruction and waits for the interrupt. Upon return from interrupt, it proceeds with the calculation of peak and rms values.

Once the peak and rms values have been calculated, all that remains is to display the results. The display routine masks the hexadecimal data in groups of 4 bits each.

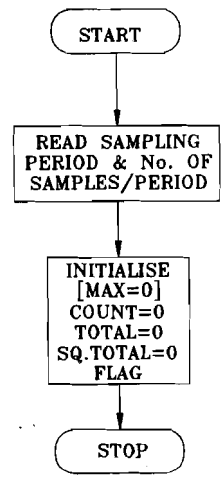
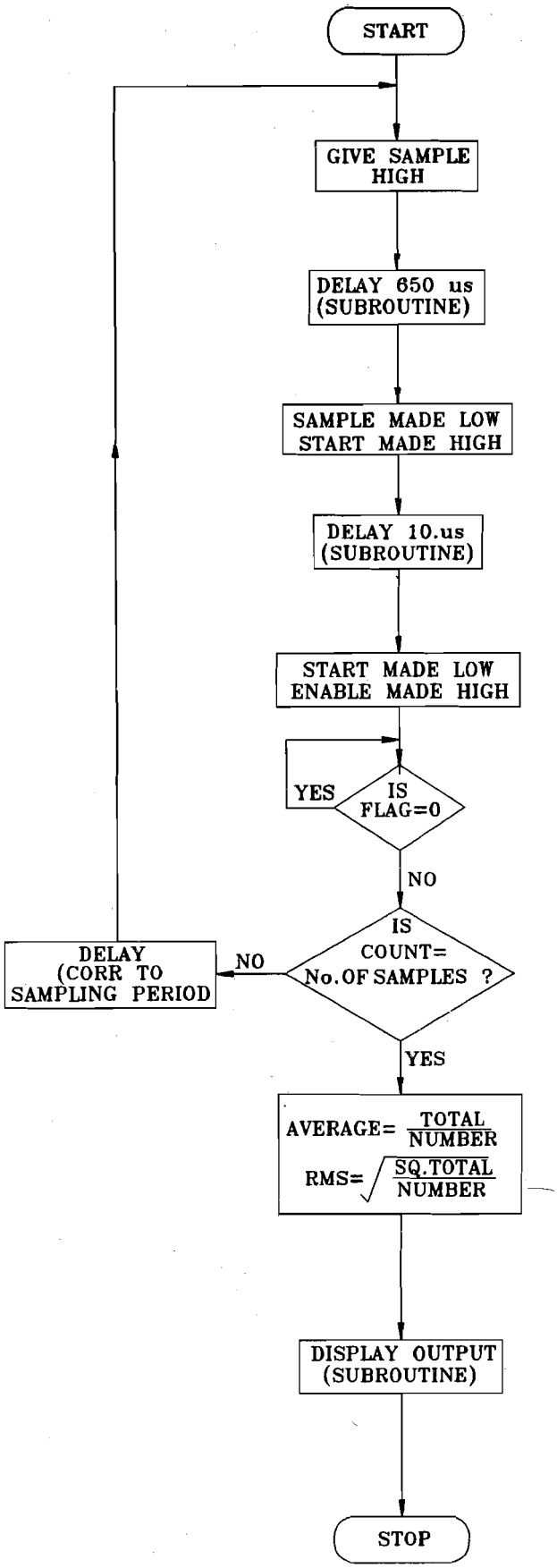


Figure 19.16 : Input and Initialization Subroutine

The masked data are sent out through port *A* of the 8755 so that the data will be present before display registers are clocked. Once the data are stable, the masked data are ORed with control bits to clock the display registers. Four 9356 latch/decoders are used for the display.

The flow charts are given in Figures 19.16 to 19.19 and a listing of the programs follows.



Note : FLAG is changed in the interrupt service routine.

Figure 19.17 : Data Processing Routine (Main Program)

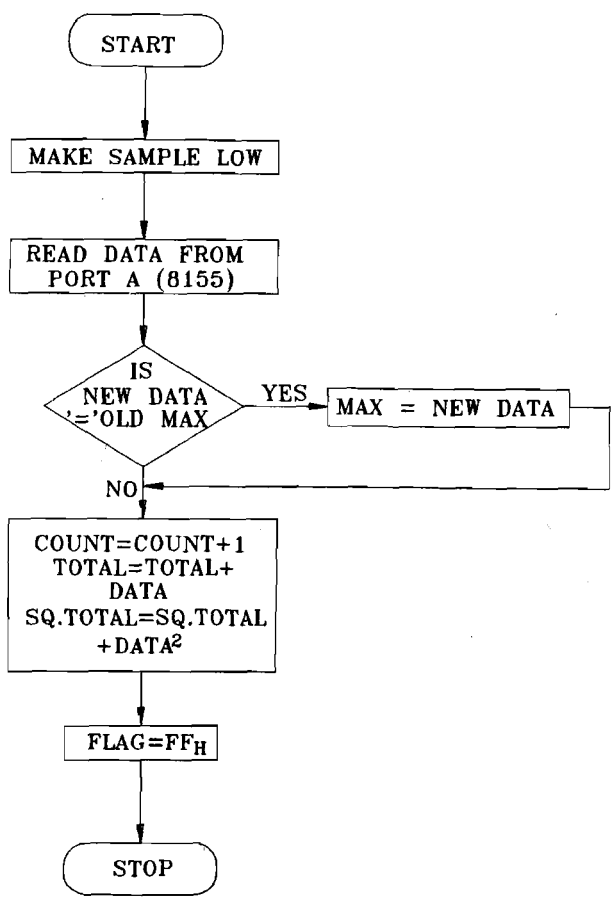


Figure 19.18 : Interrupt Service Routine

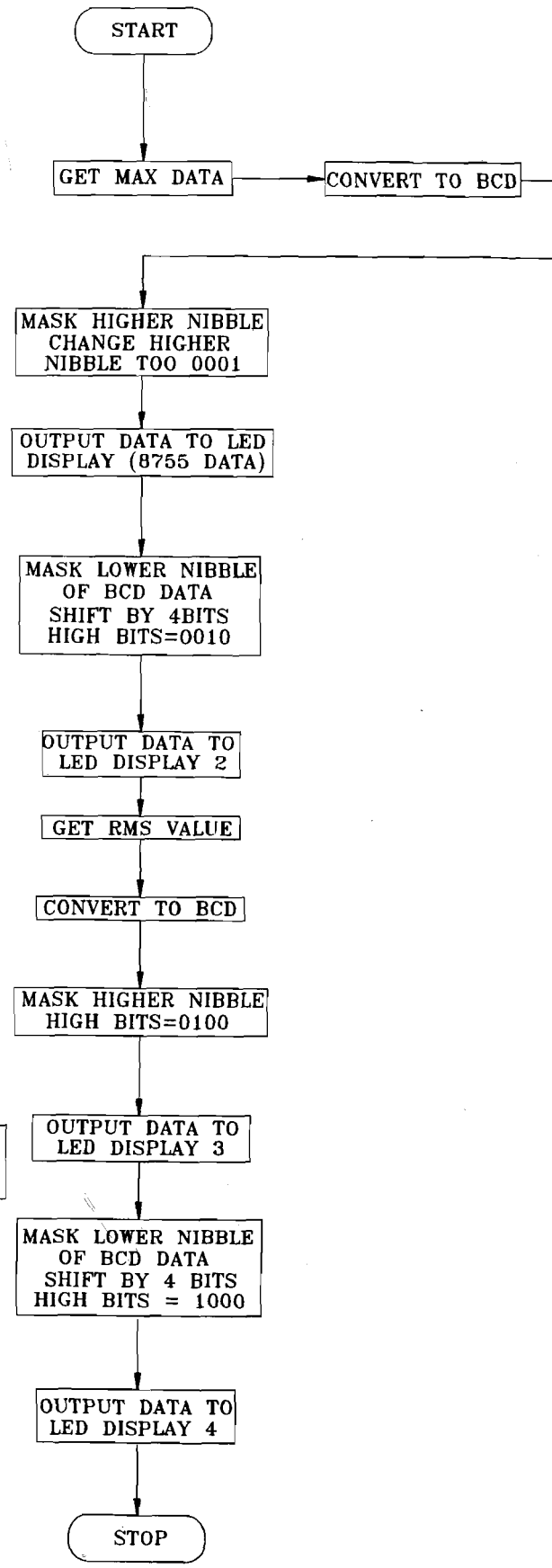


Figure 19.19 : Output Display Routines
Note : LED displays 1&2 display maximum magnitute in BCD. LED displays 3&4 display RMS value in BCD.

Program code

```
        LXI SP, 20C2H       ;   initialize stack pointer
        CALL INPUT          ;   call subroutine to get
                                DELAY and NUMSAMPLE and
                                initialise MAX
        MVI A, 02H
        OUT 08H             ;   configure 8156
        MVI A, 00H
        OUT 0AH             ;   clear all Port B signals
AGAIN : MVI A, 01H
        OUT 0AH             ;   set pin PB0 (sample) HIGH
        CALL DLY1           ;   Give delay of 650 μ secs
        MVI A, 00H
        OUT 0AH             ;   make SAMPLE signal low
        MVI A, 04H
        OUT 0AH             ;   make START (of conversion) signal
                                high (Pin PB2)
        CALL DLY2           ;   Give delay of 10 μs
        MVI A, 00H
        OUT 0AH             ;   bring START signal down
        MVI A, 02H
        OUT 0AH             ;   make Pin PB1 INT ENABLE
                                signal high
        MVI B, 00H          ;   B is a flag which gets altered in the
                                interrupt service routine
BACK :  CMP B, 00H
        JZ BACK             ;   wait for interrupt
        LXI B, NUMSAMPLE
        LDAX B              ;   load B with contents of NUMSAMPLE
                                (total no. of samples/period)
        LXI H, COUNT
        CMP M               ;   check if count = tot.no. of samples
                                required
        JZ OVER
        CALL DLY3           ;   Give delay equal to sampling period
        JMP AGAIN           ;   if there are more samples to be read in,
                                jump back to start.
OVER :  LXI B, TOTAL        ;   To calculate average, we divide the
        LDAX B              ;   sum of the magnitudes by number
        MOV D, A            ;   of samples (stored in mem locations
        LXI B, NUMSAMPLE    :   TOTAL & NUMSAMPLE resply.)
        LDAX B              ;
        CALL DIVIDE         ;   DIVIDE subroutine takes in inputs from
        LXI B, AVERAGE      ;   registers A and D and gives the quotient in
        STAX B              ;   A and the remainder in D
                            ;   Answer is stored in AVERAGE
```

```
        LXI B, SQTOTAL      ;   To calculate rms, we calculate the
        LDAX B              ;   mean square and then take its
        MOV D,A             ;   square root; subroutine SQROOT
        LXI B, NUMSAMPLE        takes in accumulator value and
        LDAX B              ;   returns its root in A
        CALL DIVIDE         ;
        CALL SQROOT
        LXI B, RMS
        STAX B
        CALL OUTPUT         ;   output subroutine displays all results
        RST
```

Interrupt Service Routine

```
        PUSH PSW            ;   store existing status
        PUSH H
        MVI A, 00H
        OUT 0AH             ;   set ENABLE low
        IN 09H              ;   Read in data from A/D
        MOV B,A
        LXI H,MAX           ;   compare data read in with the max
        CMP M               ;   stored in MAX; if the new data is
        JC PROCEED          ;   higher, update MAX
        MOV M,A
PROCEED : LXI H, TOTAL
        ADD M
        MOV M,A             ;   update the total
        LXI H,COUNT         ;   update the count
        INC M
        MOV A, B            ;   update the mean square value
        CALL SQUARE
        LXI H, SQTOTAL
        ADD M
        MOV M,A
        MVI B, FFH          ;   set flag
        RET                 ;   return
DLY1 :  PUSHB               ;
        MVI C, 59H          ;   Tot.no.
back1 : DCR C               ;   of clock cycles
        JNZ back 1          ;   add up to approx 1300;
        POP  B              ;   assuming 5 MHz clock
        RET                 ;   time delay = 650 µs
DLY2 :  NOP                 ;
        NOP                 ;   Delay of 10 µs approx
        NOP                 ;   including CALL & RET
```

```
        NOP             ;
        NOP             ;
        RET             ;
DLY3 :  PUSH H          ;   This subroutines gives
        LXI H, DELAY    ;   delay of DELAY x 0.1 ms
back3 : CALL DLY4       ;
        DCR M           ;
        JNZ Back3       ;
        POP H
        RET
DLY4 :  PUSH C          ;   DLY 4 gives a delay of 0.1 msec. approx
        MVI C, DE_H
back4 : DCR C
        NOP
        NOP
        JNZ BACK4
        POP C
        RET
OUTPUT subroutine
        PUSH PSW
        PUSH B
        PUSH H
        MVI A, FF_H         configure port A of 8755 as output port
        OUT 02_H        ;
        LXI H, MAX
        MOV A, M
        DAA             ;   to convert it to BCD
        MOV B, A
        ANI OF_H        ;   mask higher nibble
        ORI 10_H        ;   set Pin D4 to 1 to enable the reqd.
                            LED display
        OUT 00_H        ;   port A of 8755
        MOV A, B
        RRC
        RRC             ;   four rotate operations
        RRC             ;   interchange the lower & higher
                            nibbles
        RRC
        ANI OF_H        ;   similar to previous case (mask higher
                            nibble)
        ORI 20_H        ;   Pin D5 to 1 to enable LED display 2
        OUT 00_H
        MXI H, RMS      ;   To display rms value
        MOV A, M
        DAA             ;   conversion to BCD
```

```
MOV B, A                    similar operations as above
ANI OF_H
ORI 30_H
MOV A, B
RRC
RRC
RRC
RRC
ANI O4_H
OUT 00_H
POP H
POP B
POP PSW
RET
```

Note:

— LED displays 1 & 2 display max value in BCD

— LED displays 3 & 4 display rms value in BCD

19.7 SUMMARY

In this Unit, we have learnt the techniques of interfacing peripherals such as memory and input/output devices to the microprocessor. We have studied the various design steps in memory and I/O interfacing along with necessary circuitry. We were also introduced to some advanced concepts in microprocessor interfacing such as Interrupts, Serial I/O and Direct Memory Access. Finally, we understood the issues involved in the design of microprocessor based systems and studied a typical application example.

19.8 ANSWERS TO SAQs

SAQ 1

2 K-bytes = 0800_H bytes

Starting address 7000_H

Ending address $7000_H + 8000_H - 1 = 77FF_H$

Since each of the devices given is 1 K x 4 bits, we need 4 chips to get a total size of 2 K-bytes because

$$\frac{2\,K \times 8}{1\,K \times 4} = 4$$

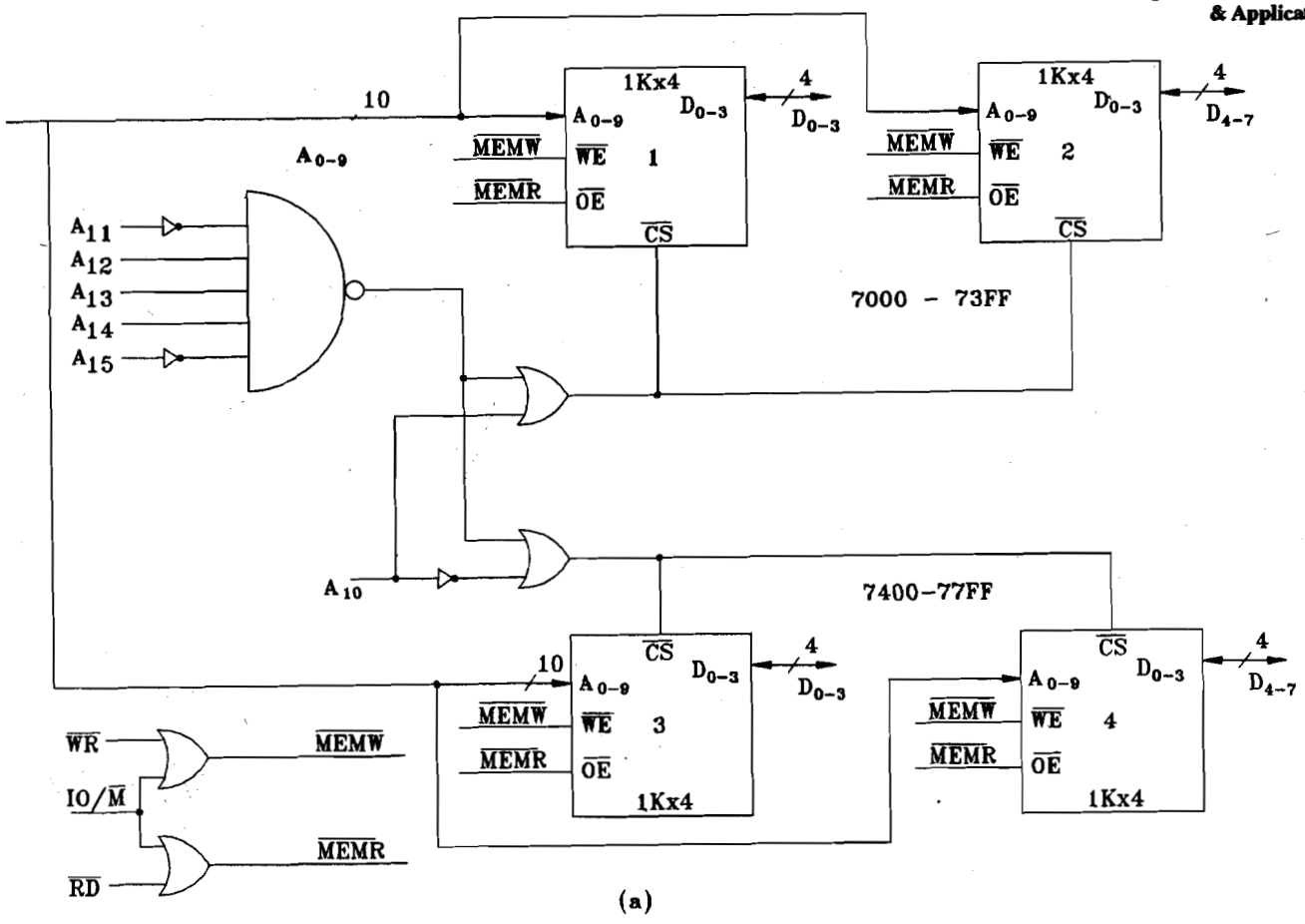
Each 1 K memory requires 10 bits address and the total memory of 2 K requires 11 bits. That is, we will use address lines A_0-A_9 for each device to select one of the 1 K locations and line A_{10} to select between the two subranges as shown below:

Subrange	A_{10}
7000 - 73FF	0
7400 - 77FF	1

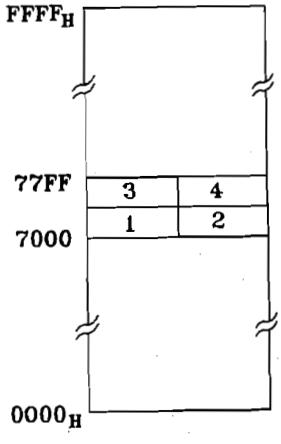
The remaining address lines A_{11} to A_{15} should have the following values to select the addresses given in the problem :

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}
0	1	1	1	0

They can be combined in a NAND gate to generate a select signal which can then be combined with the required level of A_{10} to form the $\overline{CS}$ signals for the two subranges.



(a)



(b)

Figure for Answer to SAQ 1
(a) Circuit (b) Memory map

Each 1 K memory device has only four data lines since its size is 1 K x 4 bits. To get 8 data lines for each subrange, we should use two devices in parallel, the first one connected to data lines D_{0-3} and the second one to the data lines D_{4-7} of the data bus of the 8085. The circuit is shown above.

SAQ 2

A BCD display has four inputs as shown in Figure for Answer to SAQ 2(a). When a binary coded decimal value is given to these inputs, the corresponding decimal digit appears on the display. Two such displays can be connected to the 8 data lines of the output port as shown in Figure for Answer to SAQ 2(b).

The port consists of an octal latch/buffer. Information to be displayed is latched on from the microprocessor by clocking the latch and by enabling the buffer permanently, the display in maintained continuously. The design of the port along with the circuit to decode port 80_H is shown in the Figure for Answer to SAQ 2(c).

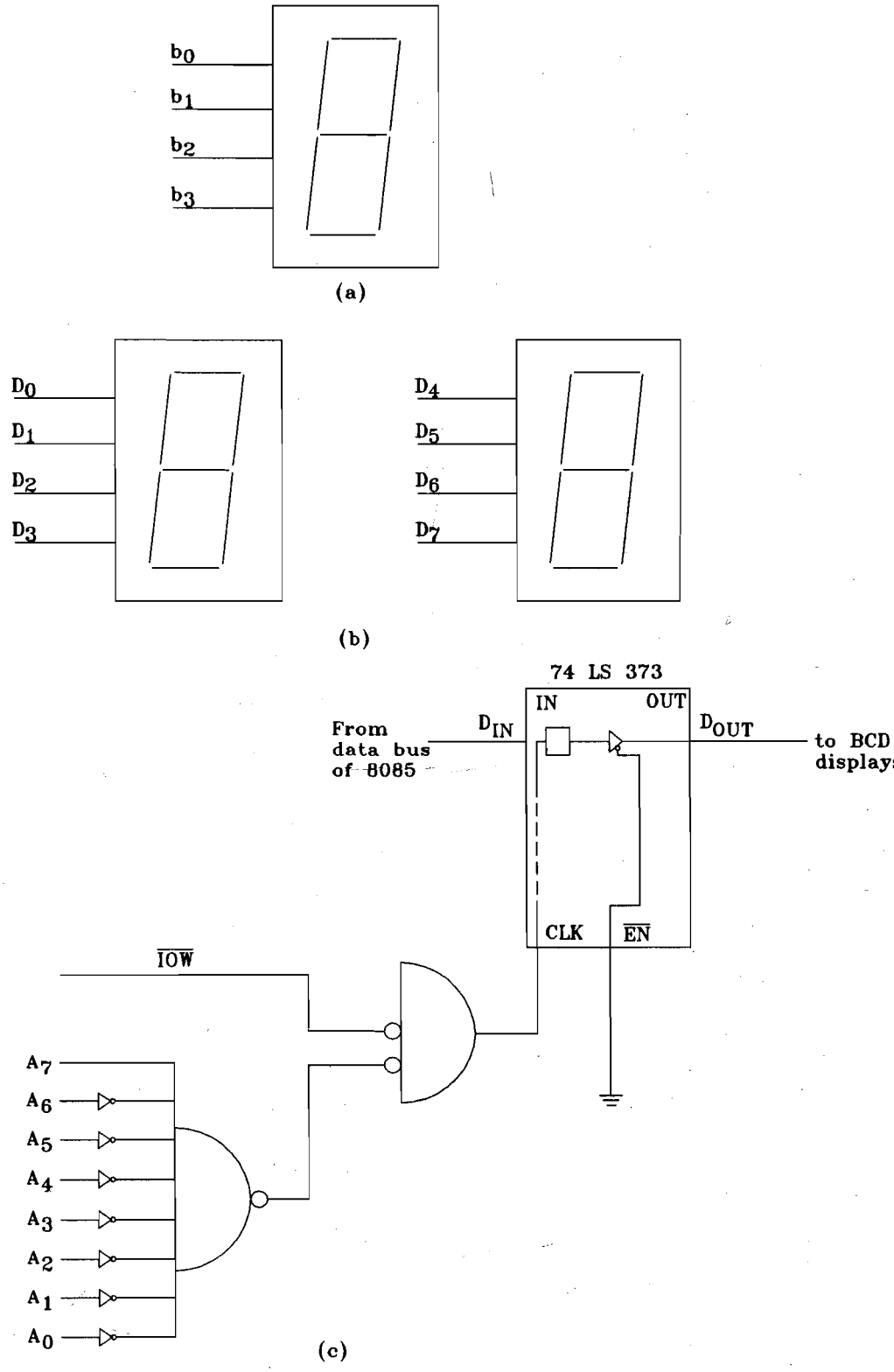


Figure for Answer to SAQ 2
(a) A BCD display (b) The output device (c) Output port along with decoding circuit

BLOCK 4 FURTHER READING

1. Schaum's outline series : Digital Principles, 2nd Ed., R.L., Tokheim, McGraw Hill, 1988.
2. Digital Principles and Applications, A.P. Malvino and D.P. Leach, Tata McGraw Hill, 1975
3. Digital Logic and Computer Design, M. Mano, Prentice-Hall India, 1979.
4. Microprocessor Architecture, Programming and Applications with the 8085/8080A, R.S. Gaonkar, Wiley Eastern, 1987.
5. Microprocessors and Programmed Logic, 2nd Ed., K.L. Short, Prentice-Hall, 1987.
6. Microprocessors: Principles and Applications, Ajit Pal, Tata McGraw Hill, 1990.
7. Microcomputer Theory and Applications, Mohamed Rafiquzzammen, John Wiley, 1982.
8. Electronic Instrumentation and Measurement Techniques, W.D. Cooper and A.D. Helfrick, Prentice Hall of India, 1986.